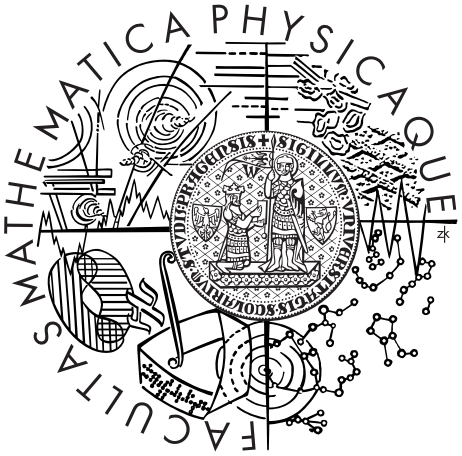
Charles University in Prague

Faculty of Mathematics and Physics

BACHELOR'S THESIS



David Jurenka

Upper Bounds for (k, s)-SAT

Department of Applied Mathematics

Supervisor: Stefan Szeider, Department of Computer Science, University of Durham

Study programme: Mathematics

Specialization: General Mathematics

2011

The present thesis was written during the academic year 2008/2009, which I spent at the University of Durham as an LLP/Erasmus student. I would like to thank my supervisor Stefan Szeider for introducing me to the topic of the thesis and for his kind guidance along the way. I also wish to express my gratitude to the Ministry of Education, Youth and Sports of the Czech Republic and to the Education, Audiovisual and Culture Executive Agency of the European Commission for the financial support of my stay.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 31. července 2011

David Jurenka

Název práce: Horní odhady pro (k, s)-SAT
Autor: David Jurenka
Katedra: Katedra aplikované matematiky
Vedoucí bakalářské práce: Stefan Szeider, Department of Computer Science, University of Durham

Abstrakt: Při studiu problému (k, s)-SAT narážíme na funkci $f : \mathbf{N} \to \mathbf{N}$ vymezující hranici mezi triviálními a **NP**-těžkými variantami tohoto problému. Nicméně přesné hodnoty $f(k)$ pro $k > 4$ nejsou známy.

Tato práce se zabývá funkcí f, hledáním jejích horních odhadů pro malé hodnoty k a generováním minimálních formulí (k, s)-CNF dokládajících tyto odhady. Horní odhady jsou získány implementací heuristiky nedávno popsané v časopisecké literatuře. Výpočet je následně paralelizován pro využití potenciálu počítačových clusterů pro HPC.

Dále je dokázáno několik teoretických výsledků. Vyvrátíme jeden nedávný dolní odhad pro f, zavedeme dolní odhad velikosti vynucujících formulí pro (k, s)-SAT a zlepšíme stávající faktor neaproximovatelnosti problému MAX-$(3, 4)$--SAT.

Klíčová slova: (k, s)-SAT, kritická funkce, minimální vynucující formule, 2-barvení hypergrafů, neaproximovatelnost

Title: Upper Bounds for (k, s)-SAT
Author: David Jurenka
Department: Department of Applied Mathematics
Supervisor: Stefan Szeider, Department of Computer Science, University of Durham

Abstract: While studying the (k, s)-SAT problem, one encounters a function $f : \mathbf{N} \to \mathbf{N}$ demarcating the boundary between trivial and **NP**-hard variants of this problem. However, the exact values of $f(k)$, for $k > 4$, are not known.

In this paper, we investigate the function f, find its upper bounds for small values of k and generate minimal instances of (k, s)-CNF formulas certifying these bounds. The upper bounds are obtained by implementing a recently published heuristic. The computation is subsequently parallelized in order to exploit the possibilities of high-performance computing grids.

In addition, a number of related theoretical results are obtained. We disprove a recent lower bound for f, establish a lower bound on the size of (k,s)-SAT enforcers and improve on the current factor for inapproximability of MAX-$(3,4)$--SAT.

Keywords: (k,s)-SAT, Critical function, Minimum enforcers, Hypergraph 2-colouring, Inapproximability

Contents

1 Introduction

The satisfiability of propositional-logic formulas in conjunctive normal form (or SAT for short) was the first decision problem shown to be **NP**-complete. Cook (1971) in his seminal paper proved that even a weaker problem, 3-SAT, SAT where we consider only formulas with clauses of length 3, is a complete problem within the **NP** class. It is not difficult to generalize this result by concluding that k-SAT, for $k \geq 3$, is **NP**-complete as well. On the other hand, 2-SAT can be decided in polynomial time by the resolution method, and 1-SAT is a very easy decision problem. In further text, we will refer to instances of k-SAT as to k-CNF formulas.

Tovey (1984) investigated the properties of 3-SAT and imposed an additional restriction on the formulas considered, namely the maximal number of occurrences each variable can have in the formula. By applying the classic Hall's marriage theorem, Tovey showed that every 3-CNF formula is satisfiable, provided the number of occurrences of every variable is limited by three. Hence, the related decision problem is entirely trivial. However, he was able to reduce the general 3-SAT problem to instances where every variable had at most only 4 occurrences. Therefore, if the limit on the number of occurrences is greater than three, 3-SAT remains **NP**-complete.

Kratochvíl, Savický and Tuza (1993) further generalized Tovey's results. They introduced the notion of (k, s)-SAT, which denotes the decision problem of satisfiability of formulas in conjunctive normal form where every clause consists of exactly k literals and every variable has at most s occurrences (positive or negative). A related problem to this is $(k, = s)$-SAT, where every clause has to have *exactly* s occurrences. In addition to this, we shall also use the symbol (k, ∞)-SAT to refer to the decision problem for CNF formulas with clauses of length k and with no limit on the number of occurrences of variables. (k, ∞)-SAT is essentially identical to k-SAT; we shall use this notation only occasionally to emphasize the absence of restriction on the number of occurrences of variables. In their paper, Kratochvíl et al. showed that a situation similar to the one observed by Tovey in the case of 3-SAT occurs in k-SAT for any $k \geq 3$. If k is fixed, the problem of (k, s)-SAT is trivial (every instance is a YES instance) for small values of s. However, if we gradually increment s, at some point we reach a threshold, and the problem becomes **NP**-complete. There is no transition stage: the complexity suddenly jumps from complete triviality to the worst case possible within the class of SAT. Let us denote by $f(k)$ the largest value of s such that (k, s)-SAT is still trivial; $f(k) + 1$ is then the smallest value of parameter s

for which (k, s)-SAT is already **NP**-hard. This defines a function that we will call the *critical function* f of (k, s)-SAT.

There is no known analytic description of f; only upper and lower bounds have been found, and it is not even known whether f is computable. This paper focuses on an algorithmic approach to obtaining upper bounds of $f(k)$ for small values of k, as described by Hoory and Szeider (2005). The basic idea of this approach is simple: We know that $(k, f(k))$-SAT contains only satisfiable formulas, hence if we manage to find an unsatisfiable k-CNF formula with at most s occurrences of every variable, we can conclude that surely $s > f(k)$, which represents our upper bound. However, the search within the class of (k, s)-CNF formulas cannot be exhaustive, and a suitable heuristic has to be employed instead. Details on the heuristic will be given in chapter 3.

As a part of this project, an implementation of the heuristic conceived by Hoory and Szeider has been developed, and by running the programme upper bounds of $f(k)$ for small values k have been obtained. Subsequently, the algorithm has been further adjusted to look not only for unsatisfiable formulas, but also for small enforcers in (k, s)-SAT. An *enforcer for variable* x is a formula that is satisfiable, but all its satisfying truth-value assignments set $x = 1$; we then say that such a formula enforces x to be true. Enforcers can be used to easily construct unsatisfiable formulas or to increase the size of an arbitrary clause without influencing the satisfiability of the whole formula. In this case, a new variable is added to the deficient clause, and this variable is then forced to be false by appending a corresponding enforcer to the formula. The new variable will then have one occurrence in the original part of the formula and also a certain number of occurrences in the enforcer. If the total number of occurrences must not exceed s (as it is the case in (k, s)-SAT), we require that the enforcer contain at most $s - 1$ occurrences of the variable being enforced, in addition to the standard (k, s)-SAT requirement that every variable occur at most s times. Among other things, this technique is very useful in inapproximability proofs. As an example, the smallest possible $(3, 4)$-SAT enforcer will be presented in this paper, and the current best factor of inapproximability of the maximization variant of this decision problem will be improved upon.

The original heuristic did not work with formulas directly. Instead of an unsatisfiable formula itself, it produced only a certificate of its existence. A routine has been added to actually generate these formulas and also related enforcers.

Because of the nature of the search algorithm, the computation becomes extremely resource-intensive already for relatively small inputs. Therefore, the al-

gorithm has been additionally fully parallelized so that it could be run on high-performance computing clusters.

Several theoretical results have also been obtained as part of this project. A smaller $(3,4)$-SAT enforcer has been found than the smallest one currently known (which also happens to be of the same size as the one produced by our enforcer search heuristic). This resulted in an improvement of the inapproximability factor for MAX-$(3,4)$-SAT. In order to show the minimality of this enforcer, a general lower bound on the size of (k,s)-CNF enforcer is proved. The present paper also touches upon the connection between (k,s)-SAT and the problem of 2-colourability of hypergraphs and disproves some recent results on the lower bounds for (k,s)-SAT, obtained by techniques used in the hypergraph-colourability theory.

2 Related work

2.1 Theoretical bounds

The main focus of the present project is the critical function f of (k,s)-SAT and its behaviour, mainly the bounds on its growth, both lower and upper, direct and asymptotic. Basic bounds were already given in the primal paper by Kratochvíl et al. By using classic techniques from hypergraph 2-colouring (mainly the celebrated Lovász local lemma), they showed that $f(k) \geq \lfloor \frac{2^k}{ek} \rfloor$, for every $k \geq 1$, which continues to be the best known lower bound until the present day. A better but erroneous bound was obtained recently, as it will be further discussed later. In addition, better lower bounds are known for small values of k. These were obtained either by a direct application of Hall's theorem (Kratochvíl et al.; 1993), by a series of involved formula transformations (Dubois; 1990), or by re-employing some more advanced ideas and techniques originally devised for hypergraph 2-colouring (Berman, Karpinski and Scott; 2003).

Regarding the upper bound, Kratochvíl et al. showed by a very straightforward argument that $f(k) = O(2^k)$, which together with the lower bound confines f into the region of exponential growth. This bound was later improved by Savický and Sgall (2000) to $f(k) \leq \frac{2^k}{k^\alpha} - 1$, where $\alpha = \log_3 4 - 1 \approx 0.26$. The best upper bound known at the present moment was obtained by Hoory and Szeider (2006), who showed that $f(k) = O(\frac{2^k \log k}{k})$, which is tight up to a $\log k$ factor with the best lower bound.

A recent paper by Gong and Xu (2006) introduced an erroneous lower bound $f(k) = \Omega(\sqrt{\frac{k}{\ln k}}\frac{2^k}{k})$, which even turns out to be asymptotically larger than the upper bound by Hoory and Szeider by a factor of $\sqrt{\frac{k}{\log^3 k}}$. This paper is yet again based on a hypergraph colouring technique, and this relation in general together with the result in particular certainly deserve further discussion.

2.2 Relation to hypergraph 2-colourability

Many of the results on (k,s)-SAT were obtained using techniques and methods originally developed in context of studying the 2-colourability of hypergraphs. A *hypergraph* is an ordered pair $H = (V,E)$, where V is a non-empty set (the vertices) and E is a set of non-empty subsets of V (the edges). Hypergraph is called k-uniform if all its edges have cardinality k. Based on this definition, we may consider standard graphs as 2-uniform hypergraphs. Furthermore, H is said to be c-colourable if there is a function $V \to \{1,2,\ldots,c\}$ such that no edge is monochromatic, i.e., there exists no edge whose all vertices received the same value. We can see that this directly generalizes the notion of chromatic number and colourability of (ordinary) graphs. However, in contrast with graphs, deciding whether a given hypergraph is 2-colourable is an **NP**-complete problem, even if all edges have cardinality of at most 3. Hypergraph 2-colourability is one of the most prominent problems in combinatorics, and it has been intensively studied since the beginning of the past century (the term 'property B' for 2-colourability is also traditionally used; it was introduced by Miller (1937) in honour of Bernstein (1908), who first studied this problem). Fundamental progress on this issue was made by Erdős (1963), one of whose results being that any k-uniform hypergraph with less than 2^{k-1} edges is 2-colourable. To prove this, he used a probabilistic approach which later became a classic tool in combinatorics in general, and in satisfiability problems in particular. Erdős argued that if we colour the vertices uniformly and independently at random with two colours, then for hypergraphs with not too many edges the expected number of monochromatic edges is less than 1, which implies that there has to be a 2-colouring. A similar argument will be presented in the proof of a lemma in chapter 4. Future contribution was made by Lovász (who in a co-authored paper with Erdős (1975) introduced the celebrated Local lemma), Spencer and Beck.

From our standpoint, it is important that the question of 2-colourability of k-uniform hypergraphs is very closely related to (k,s)-SAT. We can view a k-CNF formula as a hypergraph where vertices are the variables appearing in the for-

mula, and the problem of satisfiability translates into the task to colour every vertex with either TRUE or FALSE. At this point it is important to mention another basic notion in the theory of hypergraphs, the overlap. The *overlap* of an edge is the number of edges it intersects (including itself). The overall overlap of a hypergraph is then the maximum overlap of its edges. The relation between the size of the overlap and 2-colourability has been intensely studied, and many probabilistic colouring techniques exploiting the properties of the overlap have been devised. The importance of overlap stems from the fact that the success of colouring of certain edge under a random colour assignment is independent of success of colouring of edges that it does not overlap. It was also one of the original applications of the Lovász local lemma to show that any k-uniform hypergraph with overlap less than $\frac{2^{n-1}}{e}$ is 2-colourable.

This consideration of overlap further translates into (k, s)-SAT: If we bound the number of occurrences of variables in a k-CNF formula by s, we automatically bound the number of clauses each clause can overlap (i.e., contain a common variable) by $k \cdot (s - 1)$. Notice also the similarity between the just stated lower bound for hypergraph 2-colourability and the lower bound for (k, s)-SAT by Kratochvíl et al., the only difference being an extra factor of k in the denominator. However, the overall correspondence between satisfiability of (k, s)-CNF formulas and property B of k-uniform hypergraphs with bounded overlap is far from perfect: Firstly, in k-SAT we want each clause to contain at least one true literal, it does not matter if all of them are true, which is in sharp contrast to hypergraph 2-colourability, where we want every edge to be non-monochromatic. However, the main difference lies in the fact that variables in CNF formulas can appear in two contexts (as positive or negative literals). A k-uniform hypergraph on n vertices can have up to $\binom{n}{k}$ different edges, whereas a k-CNF formula with n variables can have up to $2^k\binom{n}{k}$ clauses. And it is this difference which turns out to be of crucial importance. Despite this, numerous techniques developed for hypergraphs were subsequently adjusted for application in satisfiability theory. And indeed, as we already mentioned, the majority of results on (k, s)-SAT were obtained in this fashion.

A recent paper by Radhakrishnan and Srinivasan (2000) significantly improved on the random-colouring approach for hypergraphs. The authors define the *method of alteration*, which is an elaboration of the refined probabilistic method, as developed mainly in the work of Beck and Spencer. In the alteration method, one starts with a random colour assignment. Such a colouring is usually far from perfect, and therefore the assignment is then reprocessed so that some of

its 'glitches' get corrected. The two most popular approaches are either to proceed completely deterministically, or to correct the glitches again by random and mutually independent processes. Radhakrishnan and Srinivasan introduced a synergic technique called the *slow recolouring*, which combines the best of both these approaches. The vertices are processed separately one by one, and their colours are changed only if the vertex lies in a monochromatic edge (this particular glitch thus gets rectified), but only if the vertex is in addition 'qualified' for being altered. This qualification is based upon the value of another random variable assigned uniformly and independently to every vertex. This method leads to a significantly better lower bound for 2-colourability of uniform hypergraphs with bounded overlaps. The authors first show that for a sufficiently large k any k-uniform hypergraph with at most $0.7\sqrt{\frac{k}{\ln k}}2^k$ edges is 2-colourable, and subsequently they generalize the result to a 'local' version, using the Lovász local lemma. By 'localizing' the result we mean that they prove a sufficient condition for 2-colourability without referring to the total number of edges but only to the overlap, which can be regarded as a local property of the hypergraph. The bound states that for sufficiently large k any k-uniform hypergraph with overlap at most $0.17\sqrt{\frac{k}{\ln k}}2^k$ is 2-colourable. Indeed, this can be easily seen as a strengthening of the bound related to the number of edges because it is always the case that the overlap is at most the total number of edges.

There has been a significant level of hope that this method could be adjusted to the problem of (k,s)-SAT and that it could lead to an improvement of the already somewhat venerable lower bound of Kratochvíl et al. This belief was also expressed explicitly in the concluding remarks of Berman et al. (2003).

This challenge was taken up by Gong and Xu (2006), who in their conference paper (which was subsequently published as the opening article of the first issue of a new scientific journal, the Journal of Information and Computing Science) literally repeated the procedure of Radhakrishnan and Srinivasan with only a few small alteration to accommodate the apparent differences of assigning a truth value to a CNF formula from constructing a 2-colouring for a hypergraph, and thereby they obtained a set of lower bounds for (k,s)-SAT, analogous to those mentioned above for hypergraphs. The authors claim that every (k,∞)-CNF formula with less than $0.58\sqrt{\frac{k}{\ln k}}2^k$ clauses is satisfiable. Additionally, by localizing this result using the Lovász local lemma they obtained an asymptotic lower bound for the critical function f, stating that $f(k) = \Omega(\sqrt{\frac{k}{\ln k}}\frac{2^k}{k})$. (It is worth noticing that—exactly as in the case of the lower bound by Kratochvíl

et al.—this bound differs from the underlying bound for uniform hypergraph by a factor of $\frac{1}{k}$.) However, as it was mentioned already, this lower bound is not consistent with the current best upper bound by Hoory and Szeider, and it is actually not difficult to produce direct counterexamples for these estimates: Since $\lim_{k\to\infty} \sqrt{\frac{k}{\ln k}} = \infty$, we could conclude from the lower bound regarding the number of clauses that for sufficiently large k, any k-CNF formula with less than $2 \cdot 2^k$ clauses is satisfiable. However, it is very easy to see that if we take k variables and all possible clauses containing these variables, we get a formula of size 2^k (usually called the complete formula) that does not admit any satisfying truth-value assignment. The lower bound for the critical function was obtained from a related proposition about satisfiability with respect to overlap, stating that for any $k \geq 2$ every (k,∞)-CNF formula with overlap less than $0.1\sqrt{\frac{k}{\ln k}}2^k$ is satisfiable. For any (k,s)-CNF formula the relation between its overlap d and s is given by the inequality $d \leq k \cdot (s-1)$. From this we can obtain the lower bound on f. However, the underlying bound regarding the overlap is again flawed: it is asymptotically strictly greater than 2^k, yet again the complete formulas on k variables are unsatisfiable and have overlap of exactly 2^k.

3 Solution

3.1 Algorithm

The algorithm used in the solution was published by Hoory and Szeider (2005). The main idea behind the heuristic is to confine the search of an unsatisfiable (k,s)-CNF formula only to a certain subclass of CNF formulas for which an exhaustive search can be performed in a finite number of steps.

A CNF formula is called *minimal unsatisfiable* if it is unsatisfiable and removing any of its clauses makes it satisfiable. We will denote the class of such formulas by MU. It is easy to see that when we search for an unsatisfiable formula in (k,s)-SAT, we can look for a formula in (k,s)-CNF $\cap$ MU. The reason is that once we have an unsatisfiable (k,s)-CNF formula, we can successively try removing its clauses until we obtain an MU formula. Moreover, removing clauses will not increase the maximal number of occurrences of variables, and thus we will stay within the class of (k,s)-CNF. From this we can see that we could redefine the critical function f as

$$f(k) = \max\{s; (k,s)\text{-CNF} \cap \text{MU} = \emptyset\}\,.$$

For a formula with m clauses and n variables we define the *deficiency* as the number $m - n$. It is known that minimal unsatisfiable formulas always contain more clauses than variables, and thus have a positive deficiency. Hence, we can partition the class MU as

$$\mathrm{MU} = \bigcup_{d=1}^{\infty} \mathrm{MU}(d)\,,$$

where $\mathrm{MU}(d)$ is the class of minimal unsatisfiable formulas with deficiency d. Davydov, Davydova and Kleine Büning (1998) showed that the formulas in $\mathrm{MU}(1)$ have an interesting recursive structure: there exists a method such that every nontrivial $\mathrm{MU}(1)$ formula can be decomposed into two smaller $\mathrm{MU}(1)$ formulas, and so on until one gets the trivial $\mathrm{MU}(1)$ formula, which consists simply of one empty clause. This recursive structure is well suited for an exhaustive search, and as it is shown in Hoory and Szeider (2005), it is actually decidable whether (k,s)-CNF $\cap\, \mathrm{MU}(1) = \emptyset$, for any given k and s. Therefore, we can define a related critical function f_1 such that

$$f_1(k) = \max\{s; (k,s)\text{-CNF} \cap \mathrm{MU}(1) = \emptyset\}\,.$$

Moreover, if we assume that for each k there is a formula in k-CNF $\cap\, \mathrm{MU}(1)$, then this function is computable. It is also clear that for every k we have $f(k) \leq f_1(k)$, and hence f_1 is an upper bound for the critical function of (k,s)-SAT. The goal of the main software deliverable of this project was to compute the values of f_1 for small values of k. For increasing k the search space grows exponentially, and the computation soon becomes intractable.

The saturation algorithm does not work directly with unsatisfiable formulas. Instead, it operates with specially devised integer sequences that represent only the relevant structure of the corresponding formulas. We start with the integer sequence representing the trivial $\mathrm{MU}(1)$ formula and combine it with itself using the recursive property of $\mathrm{MU}(1)$ to obtain a new sequence. We then proceed in stages. In each round we pick one newly inferred sequence, combine it with all previously obtained sequences and add the new results into our set of sequences inferred so far. If we infer the sequence corresponding to an unsatisfiable (k,s)-CNF formula, we halt since we got a certificate for $f_1(k) < s$. Otherwise, at one moment it will happen that we exhaust all our new sequences without inferring anything not yet present in our result set. In such case, the saturation algorithm terminates, certifying that $f_1(k) \geq s$. Therefore, in order to obtain the exact value of $f_1(k)$, we first run the algorithm with parameter k and

$s = s_0$, where s_0 is chosen based on a lower bound for f, and if we do not find an unsatisfiable formula in (k, s_0)-CNF, we successively increment s and rerun the saturation algorithm until we obtain one.

3.2 Architecture and design

The above outlined algorithm has been implemented in the Python programming language. Python was chosen for its high level of abstraction and great expression power. It also allows programmers to employ functional-programming approach in addition to the standard procedural or object-oriented one. As a functional language, Python is very strong at list manipulation, which was also the most appealing feature with respect to the project because the core of the saturation algorithm is about combining sequences of integers into other sequences, reprocessing them, comparing the resulting sequences with the older ones and finally storing them into standard or hashed arrays of results. In Python, all these procedures can be defined at a considerably high level of abstraction, using the vast variety of built-in list functions and defining custom functions using the lambda abstraction and other techniques from the functional-programming framework. This—together with Python's pseudocode-like syntax—resulted in an extremely readable and transparent code, without the need to resort to pointer manipulations and complex nested while-loop structures. On the other hand, the price for the programmer's comfort is lower performance compared to low-level languages such as C, and also a larger memory consumption, caused by the automatic memory management overhead.

The saturation in its nature is extremely resource-intensive. As a partial remedy, the algorithm has been parallelized in order to exploit the potential of high-performance computing, or HPC for short. The University of Durham is running one HPC cluster, called Hamilton. The cluster consists of nearly 300 high-performance AMD Opteron-based nodes, each running a customized version of 64-bit OpenSUSE GNU/Linux distribution. For jobs submission and management, the Grid Engine by Sun Microsystems is used. The Hamilton cluster offers a number of queues for different types of jobs. A part of the cluster is dedicated to the ordinary serial computing. The jobs from this queue are assigned to one selected node, which is then dedicated to this single task. These jobs do not truly utilize the capabilities of the cluster as a whole as they run on one single node, and the only benefit they get is the considerably high specification of the node assigned. For parallel computing, Hamilton offers two

basic approaches. The first one is the OpenMP (Open Multi-Processing) architecture, which is an application programming interface for shared-memory multiprocessing programming. There is one dedicated node with eight double-core processors and 64 GB of memory dedicated to these jobs. The second approach is MPI, and it is the only one that actually utilizes multiple nodes in parallel. MPI stands for Message Passing Interface, which is a language-independent communication protocol for parallel programming. The physical layer of communication among the nodes is facilitated by a local area networking system, usually by ordinary Ethernet or by Myrinet, which has significantly less protocol overhead and which was designed primarily for HPC clusters. Hamilton offers queues for jobs using either of these communication systems. As of May 2009, there are 337 slots for MPI over Myrinet and 110 over Ethernet. It should be also stressed out that one multiprocessor node can provide multiple computing slots at the same time, which explains why there are actually more slots in total than nodes themselves.

The MPI system was the one chosen for parallelization of the saturation algorithm. MPI is primarily used in connection with the C programming language and Fortran, but thanks to its language-independence it can be used in combination with virtually any programming language. There are a number of implementations for Python, and in this case we chose module PyPar, which is under steady development and boasts a low message-passing latency and a clean and transparent interface.

Programming in the MPI framework is similar to multiprocessing programming using forking or multithreading. Every instance is running the same piece of code; however, it has access to a method returning the total number of instances running the job and also the index of that particular instance. Based on this index and a main switch in the code, every instance then takes on the appropriate role in the computation. The inter-instance communication is facilitated by two straightforward functions: *send(message, destination_id)* and *receive(source_id)*. Moreover, these calls are blocking, and thus they serve as synchronization points for the computation as well.

The main design of the parallelized saturation algorithm is as follows: The programme requires at least two nodes to run. Node 0 is the master, the others act as slaves. As was outlined earlier, the saturation proceeds in stages, where at each stage a new sequence is picked and combined with all the previously obtained sequences. In the parallelized version every stage is further divided into rounds. At the beginning of each stage, the master node determines which

new sequence will be picked and which other sequences it will be combined with. Then it splits the inferences to be performed into packets of predefined size (this parameter can be adjusted to tune the performance of the algorithm) and assigns one packet to each of the slave nodes. It is usually the case that there are more packets than nodes, and that is also the reason why stages are divided into rounds. Once the slaves are ready with the inferences, they submit their results back to the master. The master combines the results, updates the master copy of main data structures and in the messages with the assignment for the next round it also distributes the update for slaves' data-structure copies. However, at the beginning of the computation, when we have only one sequence in our result set, it is useless for the master to distribute the computation. Therefore, the first 100 stages are always performed uniformly and independently on all nodes and only after that moment, the nodes switch into the parallel, master–slave mode.

As it was said in the introduction chapter, the algorithm was subsequently adjusted to search also for enforcers among (k, s)-CNF formulas. This required only a slight modification: instead of halting once we find a sequence corresponding to an $\mathrm{MU}(1)$ formula whose all clauses have length k, the search terminates once we derive a sequence representing an unsatisfiable formula with one to $s - 1$ clauses of length $k - 1$ and the rest of length k. We can then introduce a new variable and append it to the clauses of deficient length. Since the original formula was minimal unsatisfiable, it is easy to see that the amended formula will be an enforcer for the newly introduced variable.

Another feature which was added to the original algorithm was the ability to output an actual (k, s)-CNF unsatisfiable formula or enforcer instead of just a certificate of its existence. This is achieved by calling a new method after the end of derivation, which takes the produced certificate and for each integer sequence appearing in the derivation it generates a corresponding formula. The derivation is considerably memory-intensive since already the minimal unsatisfiable formula for $k = 7$ has millions of clauses.

3.3 Testing

The matter of testing was not a complicated issue in this project. The more complex functions were tested with small testing units containing a range of possible inputs and the corresponding expected outputs. The overall correctness of the programme was partly checked by a small script verifying the derivation

certificates produced and partly confirmed by comparing the results with those published in Hoory and Szeider (2005).

The correctness of the unsatisfiable formulas and enforcers was checked by examining their satisfiability with the programme PicoSAT, which is the winner in the category of satisfiable industrial instances of the SAT'07 SAT Solver competition and which is also directly available in the standard Debian GNU/Linux repositories.

4 Results

The following table shows the values of f_1 as computed by the saturation algorithm. For comparison, it also includes the best upper bounds obtained by purely theoretical means. The fourth and fifth columns show—where available—the number of clauses in the (k,s)-SAT enforcer and the minimal unsatisfiable formula generated by the algorithm.

Table 1: Overview of bounds obtained by the saturation algorithm

k	$f_1(k)$	theoretical upper bound	enforcer size	unsatisfiable formula size
3	3	3	6	26
4	4	4	307	767
5	7	8	946	5 196
6	11	17	39 643	172 708
7	17	35		
8	29	71		

In the following chapter we will focus on the inapproximability of MAX--$(3,4)$-SAT using small enforcers. Hence, let us now focus more closely on the $(3,4)$-SAT enforcer produced by the saturation algorithm and compare it with the enforcers obtained theoretically. When displaying a formula, we will represent a set of clauses as an array where each row lists literals of one of the clauses. By aligning the occurrences of variables we make it easy to count. This representation also helps us quickly perform resolution steps and verify that the formula is unsatisfiable or an enforcer, respectively. The $(3,4)$-CNF formula forcing x to

be true generated by our saturation algorithm has this form:

$$
\begin{array}{cccccc}
x & a & b & & & \\
x & \neg a & b & & & \\
x & & \neg b & c & & \\
 & & \neg b & \neg c & d & \\
 & & & \neg c & \neg d & e \\
 & & & \neg c & \neg d & \neg e
\end{array}
$$

To make our notation more clear, let us prove that this formula is indeed an enforcer for x. First, the formula is clearly satisfiable: setting $x = \text{TRUE}$ satisfies the first three clauses, and setting $c = \text{FALSE}$ satisfies the remaining three. However, if we assume that x is false, we can delete all its occurrences, because they are all positive. From the first two clauses we obtain b by resolution. From the third clause we then infer c, d from the fourth and e from the fifth, which leaves no way to satisfy the last clause.

From this enforcer of size six, we can easily construct an unsatisfiable $(3,4)$--CNF formula of size 19, by simply taking three disjoint copies of this enforcer, enforcing x_1, x_2 and x_3, respectively, and adding clause $(\neg x_1 \vee \neg x_2 \vee \neg x_3)$. The presented enforcer is considerably smaller than the one produced by Tovey (1984), which had size 13 and which was moreover suspected by its author to be as small as possible. Dubois (1990) later produced an enforcer consisting of only 7 clauses. Berman et al. (2003) showed the smallest $(3,4)$-SAT enforcer known at the present moment, which had length only 6 and which was then used to prove the inapproximability of MAX-$(3,4)$-SAT up to a certain small factor. The enforcer generated by the saturation algorithm has also size 6 and thus it is—though being different—on par with the best one obtained theoretically so far.

It is an interesting question, whether size 6 is actually the smallest possible. In order to prove this we will need a lower bound on the size of (k,s)-SAT enforcers. One such bound was proved as part of this project using the classic Erdős's probabilistic argument.

Lemma. *Every (k,s)-SAT enforcer consists of at least $2^k - s + 1$ clauses.*

Proof. Let $F = \{c_1, c_2, \ldots, c_m\}$ be a (k,s)-CNF formula with clauses of the form $c_i = (l_1^{(i)} \vee l_2^{(i)} \vee \cdots \vee l_k^{(i)})$, and let F be also an enforcer for x, i.e.,

$$
\begin{aligned}
c_1, c_2, \ldots, c_m &\not\models \bot\,; \\
c_1, c_2, \ldots, c_m &\models x\,,
\end{aligned}
$$

and let F be as small as possible. Let us now assume that c_m contains a negative occurrence of x, i.e., $c_m = (\neg x \vee l_2^{(m)} \vee \cdots \vee l_k^{(m)})$. Then we can infer

$$\begin{aligned} c_1, c_2, \ldots, c_{m-1}, (\neg x \vee l_2^{(m)} \vee \cdots \vee l_k^{(m)}) &\models x \\ c_1, c_2, \ldots, c_{m-1} &\models (\neg x \vee l_2^{(m)} \vee \cdots \vee l_k^{(m)}) \rightarrow x \\ c_1, c_2, \ldots, c_{m-1} &\models (x \,\&\, \neg l_2^{(m)} \,\&\, \ldots \,\&\, \neg l_k^{(m)}) \vee x \\ c_1, c_2, \ldots, c_{m-1} &\models x \end{aligned}$$

This means that we can delete from F each clause containing a negative occurrence of x and F will still be enforcing x to be true. Since we assume that F is a minimal enforcer, it must be the case that x has no negative occurrence in F.

Similarly, if we assume that x occurs positively in c_m, we get:

$$\begin{aligned} c_1, c_2, \ldots, c_{m-1}, (x \vee l_2^{(m)} \vee \cdots \vee l_k^{(m)}) &\models x \\ c_1, c_2, \ldots, c_{m-1} &\models (x \vee l_2^{(m)} \vee \cdots \vee l_k^{(m)}) \rightarrow x \\ c_1, c_2, \ldots, c_{m-1} &\models (\neg x \,\&\, \neg l_2^{(m)} \,\&\, \ldots \,\&\, \neg l_k^{(m)}) \vee x \\ c_1, c_2, \ldots, c_{m-1} &\models (\neg l_2^{(m)} \,\&\, \ldots \,\&\, \neg l_k^{(m)}) \vee x \\ c_1, c_2, \ldots, c_{m-1} &\models (l_2^{(m)} \vee \cdots \vee l_k^{(m)}) \rightarrow x \\ c_1, c_2, \ldots, c_{m-1}, (l_2^{(m)} \vee \cdots \vee l_k^{(m)}) &\models x \end{aligned}$$

From this we can conclude that after having deleted all clauses with negative occurrences of x we can also delete all positive occurrences of x from the remaining clauses. This way we get a formula F' which still implies x. However, since F' has no occurrence of x whatsoever, it is clear that F' must be unsatisfiable. F' consists of at most $s-1$ clauses of length $k-1$ (these are the clauses which previously contained a positive occurrence of x), and the remaining clauses have length k.

Let us now examine the expected number of satisfied clauses of F' under a random truth-value assignment. We will consider a probability space $(\Omega, \mathcal{S}, \mathbb{P})$, where Ω consists of all possible truth-value assignments for the variables in $var(F')$, the σ-algebra $\mathcal{S}$ is identical to the power set of Ω, and the probability measure $\mathbb{P}$ is determined by the fact that every variable is assigned the value TRUE with probability $\frac{1}{2}$ independently of the values of other variables. Thus, for every truth-value assignment $\mathbf{e} \in \Omega$ we have $\mathbb{P}(\{\mathbf{e}\}) = 2^{-|var(F')|}$. Let us

define random variables $X_i, i = 1, 2, \ldots, m$, by

$$X_i = \begin{cases} 1 & \text{if } c'_i \text{ is satisfied by } \mathbf{e}; \\ 0 & \text{otherwise.} \end{cases}$$

Then the total number of satisfied clauses c'_i of F' under the random assignment is equal to $\sum_{i=1}^{m} X_i$. For the expected value we get

$$\begin{aligned} \mathbb{E}(\sum_{i=1}^{m} X_i) &= \sum_{i=1}^{m} \mathbb{E}(X_i) = \sum_{i=1}^{m} \mathbb{P}(c'_i \text{ is satisfied}) = \\ &= \sum_{i=1}^{m} (1 - \mathbb{P}(\text{all literals of } c'_i \text{ are false})) = m - \sum_{i=1}^{m} \frac{1}{2^{size(c'_i)}}. \end{aligned}$$

Therefore, if $\sum_{i=1}^{m} 2^{-size(c'_i)} < 1$, the expected value is greater than $m - 1$, which means that there must be a truth-value assignment satisfying all m clauses. However, since we assume that F' is not satisfiable, it must be the case that $\sum_{i=1}^{m} 2^{-size(c'_i)} \geq 1$. We want to establish what the minimal value of m is such that this inequality can hold. The size of clauses c'_i is either k or $k - 1$. Since shorter clauses contribute by greater deal to the sum, we can assume that there is the maximal possible number of these shorter clauses, i.e., $s - 1$ (this translates into the assumption that our original enforcer F contains exactly $s - 1$ positive occurrences of x). Now we can obtain the final inequality for m.

$$\begin{aligned} (s-1) \cdot 2^{-(k-1)} + (m - (s-1)) \cdot 2^{-k} &\geq 1 \\ (s-1) \cdot 2 + (m - (s-1)) &\geq 2^k \\ m &\geq 2^k - s + 1 \end{aligned}$$

□

For $(3, 4)$-SAT the lemma tells us that the minimum enforcer has at least 5 clauses. It turns out that in this case the bound is sharp and that the enforcers proposed by Berman et al. (2003) and generated by the saturation algorithm are not minimum. Here we present the minimum $(3, 4)$-SAT enforcer, forcing x to be true and having only 5 clauses:

$$\begin{array}{cccc} x & \neg a & b & \\ x & & \neg b & c \\ x & a & & \neg c \\ & a & b & c \\ & \neg a & \neg b & \neg c \end{array}$$

It is worth pointing out that it also happens to be the case that all variables have the maximal possible number of occurrences: x occurs 3 times and the other variables 4 times. This formula can thus serve as a $(3, =4)$-SAT enforcer as well. We will see one application of this enforcer in the following chapter.

5 Evaluation

There are several aspects of the software to be evaluated. The soundness of the implementation was established by thorough testing and by comparing the results to those found in the literature. However, the performance of the implementation is poor. We were not able to extend the range of computed bounds beyond those already known. On the one hand, thanks to Python the code is very transparent, elegant, easy to read and maintain. Still, the price for that was a serious loss in performance. This problem was partially remedied by the parallelization which speeded up the computation by a factor close to the total number of nodes used. Yet a significant gain in performance could be achieved by rewriting the whole programme in a low-level programming language, such as C.

As a part of our investigation of small unsatisfiable formulas and enforcers, we obtained a $(3, 4)$-SAT enforcer which we proved to be as small as possible. We will now evaluate the significance of this result by improving the best currently known factor for inapproximability of MAX-$(3, 4)$-SAT and MAX-$(3, =4)$-SAT. The result is based on the work of Berman et al., who by a very technically involved construction proved the following proposition.

Proposition (Berman and Karpinski (2003); Berman et al. (2003)). *There exists a family of sets of* 2*-clauses such that each, for some* n*, consists of* $252n$ 2*-clauses and* $4n$ 1*-clauses, for which it is* **NP***-hard to distinguish between the systems where* $(252 - \varepsilon)n$ *clauses can be satisfied and systems where at most* $(251 + \varepsilon)n$ *can be satisfied (for* $\varepsilon < 1/2$*). Moreover,* $4n$ *variables in this system occur* 3 *times, while other variables have exactly* 4 *occurrences.*

We can transform a system from this proposition into an instance of MAX--$(3, =4)$-SAT in two stages. First, we increase the length of all clauses to 3 by inserting $260n$ variables that are forced to be false. Forcing them requires $260n$ enforcers, i.e., $260n \cdot 5 = 1300n$ additional clauses. Second, we need to increase the number of occurrences of $4n$ variables by one. This can be done in the following fashion: For each pair x, y of deficient variables we introduce a new

variable z and add two clauses $(x \vee z \vee \neg z)$ and $(y \vee z \vee \neg z)$. In this way we append $4n$ new clauses. Moreover, these clauses will always be satisfied, and the auxiliary variables will have the right number of occurrences, namely 4. Thus we increased the number of clauses in the system from $256n$ to $1560n$, and we can conclude.

Theorem. *There exists a family of instances of MAX-$(3,=4)$-SAT such that each, for some n, consists of $1560n$ clauses, for which it is* **NP***-hard to distinguish between the systems where $(1556-\varepsilon)n$ clauses can be satisfied and systems where at most $(1555+\varepsilon)n$ can be satisfied (for $\varepsilon < 1/2$).*

From the **NP**-hardness of this gap-problem we can directly derive the conclusion about the inapproximability of the related maximization problem.

Corollary. *It is* **NP***-hard to approximate MAX-$(3,=4)$-SAT to within any factor below $1556/1555 \approx 1.000643$.*

Please note that in the case of general MAX-$(3,4)$-SAT, we do not need to increase the number of occurrences of variables in the systems from the proposition. Therefore, we can save $4n$ additional clauses and obtain a slightly better inapproximability factor of $1552/1551 \approx 1.000645$.

6 Conclusions

The focus of this project was the (k,s)-SAT problem, its critical function and related matters. We implemented an algorithm computing an upper bound for the critical function based on the article by Hoory and Szeider. In addition to the upper-bound computation, the algorithm was adjusted to also generate small unsatisfiable formulas and enforcers. All this computation was later parallelized. We then investigated a number of related theoretical issues. A recent lower bound for the critical function based on a hypergraph-colouring technique was disproved. Next, we focused on the structure of small enforcers. An enforcer of length 5 for $(3,=4)$-SAT was presented, and by means of a general lower bound on the size of (k,s)-SAT enforcers it was proved that it was minimum. This result was then used to improve on the current inapproximability factors for MAX-$(3,=4)$-SAT and MAX-$(3,4)$-SAT.

The programming results presented in this work could be extended in part by converting the code into the C programming language. The parallelized version for high-performance cluster has a strong potential and a simple rewriting of the

code in a language with less overhead would most probably lead to extending the currently known bounds to larger values of k.

As regards the theoretical part, one very interesting question arising from this paper is whether there is a way to adjust the results of Radhakrishnan and Srinivasan to the question of (k, s)-SAT and to obtain a lower bound for the critical function based on these techniques, as it was attempted by Gong and Xu. Another possible direction for further research may be the study of the structure of minimum (k, s)-SAT enforcers. If one managed to establish in addition to the present lower bound an upper bound on the size of minimum enforcers, it would lead directly to an algorithm computing exact values of the critical function f. This would thus solve the long-open problem of the computability of this function.

References

Berman, P. and Karpinski, M. (2003). Improved approximation lower bounds on small occurrence optimization, *Electronic Colloquium on Computational Complexity (ECCC)* **10**(008).

Berman, P., Karpinski, M. and Scott, A. D. (2003). Approximation hardness and satisfiability of bounded occurrence instances of SAT, *Electronic Colloquium on Computational Complexity (ECCC)* **10**(022).

Bernstein, F. (1908). Zur Theorie der trigonometrischen Reihen, *Bericht der Königlichen Sachsischen Gesellschaft der Wissenschaften zu Leipzig* **60**: 325–338.

Cook, S. A. (1971). The complexity of theorem-proving procedures, *STOC*, ACM, pp. 151–158.

Davydov, G., Davydova, I. and Kleine Büning, H. (1998). An efficient algorithm for the minimal unsatisfiability problem for a subclass of CNF, *Annals of Mathematics and Artificial Intelligence* **23**(3–4): 229–245.

Dubois, O. (1990). On the r,s-SAT satisfiability problem and a conjecture of Tovey, *Discrete Applied Mathematics* **26**(1): 51–60.

Erdős, P. (1963). On a combinatorial problem, *Nordisk Matematisk Tidskrift* **11**: 5–10.

Erdős, P. and Lovász, L. (1975). Problems and results on 3-chromatic hypergraphs and some related questions, *in* A. Hajnal, R. Rado and V. T. Sós (eds), *Infinite and Finite Sets*, Vol. 2, North-Holland, pp. 609–627.

Gong, P. and Xu, D. (2006). New lower bound of critical function for (k,s)-SAT, *Journal of Information and Computing Science* **1**(1): 3–10.

Hoory, S. and Szeider, S. (2005). Computing unsatisfiable k-SAT instances with few occurrences per variable, *Theor. Comput. Sci.* **337**(1–3): 347–359.

Hoory, S. and Szeider, S. (2006). A note on unsatisfiable k-CNF formulas with few occurrences per variable, *SIAM J. Discrete Math.* **20**(2): 523–528.

Kratochvíl, J., Savický, P. and Tuza, Z. (1993). One more occurrence of variables makes satisfiability jump from trivial to NP-complete, *SIAM J. Comput.* **22**(1): 203–210.

Miller, E. W. (1937). On a property of families of sets, *Comptes Rendus Varsovie* **30**: 31–38.

Radhakrishnan, J. and Srinivasan, A. (2000). Improved bounds and algorithms for hypergraph 2-coloring, *Random Struct. Algorithms* **16**(1): 4–32.

Savický, P. and Sgall, J. (2000). DNF tautologies with a limited number of occurrences of every variable, *Theor. Comput. Sci.* **238**(1–2): 495–498.

Stříbrná, J. (1994). *Between combinatorics and formal logic*, Master's thesis, Faculty of Mathematics and Physics, Charles University in Prague.

Tovey, C. A. (1984). A simplified NP-complete satisfiability problem, *Discrete Applied Mathematics* **8**(1): 85–90.